ARMY RESEARCH LABORATORY

DETECT: Detection of Events in Continuous Time Toolbox: User's Guide, Examples, and Function Reference Documentation

by Vernon Lawhern, W. David Hairston, and Kay Robbins

ARL-SR-269 **June 2013**

NOTICES

Disclaimers

Army Research Laboratory

Aberdeen Proving Ground, MD 21005-5425

ARL-SR-269 **June 2013**

DETECT: Detection of Events in Continuous Time Toolbox: User's Guide, Examples, and Function Reference Documentation

Vernon Lawhern and Kay Robbins
University of Texas at San Antonio

W. David Hairston
Human Research and Engineering Directorate, ARL

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED (From - To)**
June 2013	Final	October 2012–March 2013

4. TITLE AND SUBTITLE
DETECT: Detection of Events in Continuous Time Toolbox: User's Guide, Examples, and Function Reference Documentation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Vernon Lawhern,* W. David Hairston, and Kay Robbins*

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-HRS-C
Aberdeen Proving Ground, MD 21005-5425

10. SPONSOR/MONITOR'S ACRONYM(S)
ARL-SR-269

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
*Department of Computer Science, University of Texas at San Antonio, San Antonio, TX 78249

14. ABSTRACT
DETECT (Detection of Events in Continuous Time) is a MATLAB toolbox for automated event detection in long, continuous multichannel time series. Although developed for electroencephalography (EEG), it uses a universal format that is applicable to many types of physiological time-series data or case uses benefitting from rapid, automated discrimination of specific predefined signals, and is free-standing (requiring no other plugins or packages). The primary goal is a toolbox that is simple for researchers to use, allowing them to quickly train a model on multiple classes of events, assess the accuracy of the model, and determine how closely the results agree with their own manual identification of events without requiring extensive programming knowledge or machine learning experience. Here, we provide reference documentation covering use of the DETECT toolbox, including an overview, explanations of each of the primary components and how they interact, and full help documentation for each function in the toolbox. Additionally we provide six example uses of the toolbox, including labeling trials, labeling continuous time series, manually labeling data, plotting labeled data, updating previously labeled dataset, and comparing two labeled datasets.

15. SUBJECT TERMS
Artifact, signal detection, EEG, MATLAB, toolbox

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			W. David Hairston
Unclassified	Unclassified	Unclassified	UU	56	**19b. TELEPHONE NUMBER** (*Include area code*) 410-278-5925

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

INTENTIONALLY LEFT BLANK.

1. Overview

DETECT (Detection of Events in Continuous Time) is a MATLAB[*] toolbox for automated event detection in long, continuous multichannel time series. Although developed for electroencephalography (EEG), it uses a universal format that is applicable to many types of physiological time-series data or case uses benefitting from rapid, automated discrimination of specific predefined signals, and is free-standing (requiring no other plugins or packages). The primary goal is a toolbox that is simple for researchers to use, allowing them to quickly train a model on multiple classes of events, assess the accuracy of the model, and determine how closely the results agree with their own manual identification of events without requiring extensive programming knowledge or machine learning experience. Here we provide reference documentation covering use of the DETECT toolbox, including an overview, explanations of each of the primary components and how they interact, and full help documentation for each function in the toolbox. Additionally we provide six example uses of the toolbox, including labeling trials, labeling continuous time series, manually labeling data, plotting labeled data, updating previously labeled dataset, and comparing two labeled datasets.

DETECT is a MATLAB toolbox for detecting and labeling events in epoched and continuous time series. To use DETECT, you must have examples of labeled data to create a classifier or model. Once you have created a model, you can apply it to label either epoched or continuous data. DETECT provides some utility functions specifically for manually labeling EEG data in an efficient way. This process is useful for producing labeled data to train a model for artifacts and other features.

DETECT uses autoregressive features by default, but you are free to provide your own feature functions. The toolbox includes several functions for building models of events as well as sample datasets for detecting artifact segments in EEG data.

2. Requirements

DETECT requires the following:

- MATLAB version R2011A or higher. Other versions of MATLAB may work; version R2011A and later are officially supported.
- EEGLAB version 10 or higher, if you wish to use the DETECT plotting functions.

[*]MATLAB is a trademark of Mathworks, Inc., Natick, MA.

3. Installation

The steps for installation are as follows:

1. Download the toolbox and extract the .zip file.
2. Add the extracted folder to the MATLAB Path (File → Set Path). Use the “Add with Subfolders” option.
3. Add EEGLAB and its subfolders to the MATLAB Path if you wish to use the DETECT plotting functions.

There are two versions of the toolbox available for download, depending on your installation. Currently we have versions for 64-bit Windows Vista/7 and 64-bit Linux platforms. You will need to compile LibSVM for any other installation. To recompile the toolbox, navigate to LIBSVM_DETECT/matlab and run the make.m file.

If you plan to use DETECT frequently, you may want to put commands to automatically add the DETECT folder and its subdirectories in your startup.m file located in your MATLAB startup folder.

4. Package Contents

The following chart provides detailed explanations of the package contents and functionality.

GENERAL FUNCTIONS (can be used to label any type of time series data)	
Name	**Description**
`getARfeatures.m`	Estimate autoregressive model coefficients of specified order for a three-dimension (3-D) array of input data (*channels* × *windowSize* × *windows*) and return a (*windows* × *featureSize*) array of features to be used for classification
`getModel`	Create a model or classifier based on labeled training data (*channels* × *windowSize* × *windows*)
`labelData`	Label data (*channels* × *frames*) as a function of time, reporting certainty of each label
`labelWindows`	Label windows (*channels* × *windowSize* × *windows*) based on a classification model and also return classification accuracy if ground truth labels are passed in for comparison
`compareLabels`	Compare two sets of labeled data, either from an automated labeling (from using plotLabeledData) or manual labeling (from using markEvents) or both (one set from a manual labeling and the other from an automated labeling).
EEG RELATED FUNCTIONS (depend on EEGLAB)	
`getLabels`	Convert a continuous dataset into an epoched dataset, epoching by user-highlighted regions.
`plotLabeledData`	Display results of labeling continuous using a modified EEGLAB plot window. Uses the output from labelData as input.
`plotMarkedData`	Plot a manually labeled dataset. Uses the output from markEvents as input.
`plotWindowData`	Display results of labeling windowed dataset using a modified EEGLAB plot window.
`markEvents`	Manually label data based on given categories. Can update a previously labeled dataset to add/remove events and categories. Uses a modified EEGLAB plot window.
`eegplot2`	Modified form of eegplot.m from the EEGLAB Toolbox that is used for plotting event information in DETECT.
Certainty Threshold Policies	
`thresholdPolicy`	Apply a thresholding policy. If the certainty is below a given threshold, and one of the top two possible classes is the baseline, the prediction type will be set to the baseline. No change is made if neither of the top two possible classes is the baseline. Uses output from labelData as input.

`unknownPolicy`	A thresholding policy that incorporates a new decision class of "Unknown". If the certainty is below a given threshold, and one of the top two possible classes is the baseline, the prediction type is set to the baseline. Otherwise, the prediction type is set to "Unknown." Uses output from labelData as input.
DATA	
`testing.mat`	Continuous testing data: two-dimensional (2-D) array (*channels* × *frames*)
`training.mat`	Sample training data: 3-D array (*channels* × *windowSize* × *windows*) containing the training trials for an EEG data recording using a 64-channel Biosemi recording device that includes 4 additional EOG channels (65-68). The 140 trials include 20 trials for each of 7 events.
`training-epochs.mat`	Data that has been randomly sampled from training.mat. Sampled data contains 12 trials from each class (60% random selection).
`testing-epochs.mat`	The data remaining after selection for the training-epochs.mat .Data contains 8 trials from each class (40% random selection).
`training.set`	Same as training.mat, but in EEGLAB .set format
`testing.set`	Same as testing.mat, but in EEGLAB .set format
`/SampleECG`	Folder containing Sample ECG Data obtained from the online PhysioNet Database. The data was downloaded from: http://physionet.org/physiobank/database/ltdb/
`14046_modified _training`	Epoched ECG Data containing three categories: Normal heartbeat waveforms, premature ventricular contraction (PVC) waveforms and data containing no heartbeats. This was taken from subject 14046 from MIT-BIH Long Term database
`14046_modified _testing`	Continuous ECG Data used for testing continuous detection of heartbeat waveforms.
`DETECT_ECGCode`	Sample DETECT code to analyze ECG Data
SUPPORTING LIBRARY FUNCTIONS	
`LIBSVM_DETECT`	Modified version of LibSVM that has eliminated debugging output and renamed some of the functions to avoid conflicts with MATLAB
`TSA Toolbox`	Time series analysis toolbox written by Alois Schloegl included to provide autoregressive feature functions for users without the MATLAB Signal Processing Toolbox
`Consecutive Vector Splitter (SplitVec.m)`	MATLAB function written by Bruno Luong that provides partitioning and splitting functionality for vectors of continuous elements. This is used in `compareLabels` to compare two labeled data segments. Can be found at the MATLAB File Exchange at: http://www.mathworks.com/matlabcentral/fileexchange/24255-consecutive-vector-spliter

5. EXAMPLE 1: Labeling Trials (Epoched Data)

Suppose you have epoched data consisting of a three-dimensional array (*channels* × *windowSize* × *windows*) and each trial or epoch is a *channels* × *windowSize* array. To label the data, you must have a labeled set of windows called the *training* data. The windows you wish to label are called the *testing* data. In general, the training data and testing data should not overlap.

The following MATLAB code labels the windows in the testing data based on the labeled data in the training data. Set the MATLAB Current Directory to be the directory containing the DETECT Toolbox functions and run:

```
load data/training-epochs.mat;        % load training windows + labels
model = getModel(training, training_labels); % create classifier
load data/testing-epochs.mat;         % load testing windows to be labeled
[results, accuracy] = labelWindows(testing, model, testing_labels);
```

The results structure looks like this:

```
results =

1x56 struct array with fields:
    label
    actualLabel
    certainty
    likelihoods
    prob_estimates
```

We can call the results from the first window in this array with:

```
results(1)

ans =

             label: 'None'
       actualLabel: 'None'
         certainty: 0.7793
       likelihoods: {7x1 cell}
    prob_estimates: [0.6566 0.0133 0.0141 0.0393 0.1118 0.1449 0.0201]
        labelOrder: {7x1 cell}
```

The entries in this structure are:

`results.label`	A string label indicating predicted category for the window
`results.actualLabel`	A string label indicating the actual category of the window. This is left blank if the true labels were not passed into the `labelWindows` function.

`results.certainty`	The certainty of the prediction (see below)
`results.likelihoods`	A cell array of strings denoting the categories, from most likely to least likely. The first entry in this array is the same as `results.label`.
`results.prob_estimates`	Estimated probability distribution of the classes in `results.labelOrder`.
`results.labelOrder`	Cell array of strings to identify the categories for `results.prob_estimates`. The first entry of `prob_estimates` denotes the probability of the first entry in `labelOrder`.

We calculate a certainty measure as a means to assess the confidence in the predictions. The certainty is defined as

$$\frac{P_{(1)} - P_{(2)}}{P_{(1)}},$$

where $P_{(1)}$ and $P_{(2)}$ are the first and second largest prediction probabilities for the data sample. This is a relative probability measure that quantifies the strength of the prediction: if most of the probability is concentrated in one class, this measure will be close to 1, while if the probabilities are more distributed across the classes, the measure will be close to 0.

Here, we can call `results(1).labelOrder`:

```
results(1).labelOrder

ans =

    'None'
    'Jaw Clench'
    'Jaw Movement'
    'Eye Blink'
    'Eye Left Movement'
    'Eye Up Movement'
    'Eyebrow Movement'
```

The first entry of `prob_estimates`, .6566, indicates that the probability that the data is in the class ‘None’ is .6566. The second entry, .0133 indicates the probability that the data is in the ‘Jaw Clench’ category, and so forth.

The second output, `accuracy`, is the classification accuracy. Here, the accuracy is:

```
accuracy =

   98.2143
```

By default `getModel` uses all of the channels, four-fold cross validation, and the `getARfeatures` function with parameter 2. This feature function fits an autoregressive (AR) model to each channel given. You can explicitly specify the channels to be used, the number of cross validations to perform, and your own feature function with an arbitrary number of arguments as illustrated by the following example:

```
model = getModel(training, training_labels, 1:64, 4, @getARfeatures, 2);
```

This example uses only the first 64 channels in the training data to build the classifier and four cross-validations to validate the model. We use the function `@getARfeatures` to extract autoregressive coefficients for each channel individually and concatenate all the features to form one long feature vector. The additional argument of 2 is the model order to fit. Any feature extraction function can be used here, as long as the output of the function is a matrix of size `(windows x featureSize)`with the number of features fixed. Other types of features can be used, such as spectral-based features or other features such as connectivity-based measures such as granger causality or directed coherence.

Both the training and testing data used in this example were EEG data sampled at 256 Hz using a Biosemi 64-channel Active Two System. The experiments also recorded four EOG (electrooculography) channels (channels 65–68), which are not used to build the model.

6. EXAMPLE 2: Labeling Continuous Time Series

The primary purpose of DETECT is to label continuous time series. That is, DETECT produces a time series of labels corresponding to a two-dimensional array (*channels* × *frames*) of data by sliding a window across the time series and predicting a label for each slide.

For example, suppose you want to label artifacts in your data. Pick out fixed size intervals in the time series where you recognize the artifact and label these intervals. Also choose intervals that don't contain these artifacts and label them as such (say with the label 'none'). Although DETECT does not require a balanced training set (i.e., the same number of trials for each type of feature), you should take care to provide enough training data for each type of feature that you want to classify (the more the better). Once you have labeled the training data and created a model with getModel, you can then apply labelData to produce a continuously labeled data set.

Typically, the amount of data to be labeled is very large compared with the training set, so the fact that the training set is exacted from the much larger testing test is not a cause for concern.

The size of the slide (which is given in frames or samples) is usually greater than 1. The window size must be the same as the epoch or trial size used to train the data. Figure 1 illustrates the process.

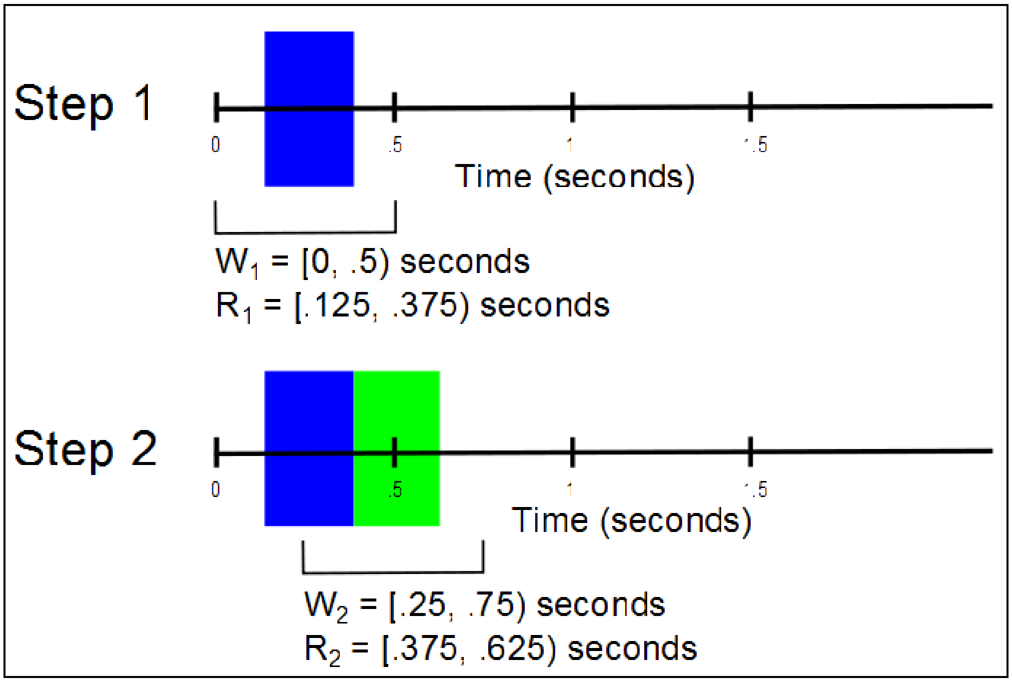


Figure 1. Illustration of the association of label with time in labeling of continuous data. The example data is sampled at 256 Hz with a window size of 0.5 s and a slide width of 0.25 s.

Figure 1 is based on the training data provided in the DETECT toolbox, using training epochs of 0.5 s (128 frames). The first window starts at time 0, and the length of the window is 0.5 s. We label data in time using the following formula:

$$R_i = [M_i - 0.5*S, M_i + 0.5*S) ,$$

where R_i is the i^{th} region of the data, M_i is the midpoint of the i^{th} window and S is the slide width, all in seconds. This procedure is performed until the end of the dataset. DETECT ignores data if the slide cannot be performed (i.e., if the slide window is 0.5 s but only 0.2 s of data remain at the end).

The following MATLAB code labels continuous testing based on the labeled data in the training data using a slide width of 0.125 s and a sampling rate of 256 Hz.

```
load data/training;                    % load the training data
load data/labels;                      % load labels for training data
model = getModel(training, labels); % create classifier using defaults
load data/testing;                     % load continuous testing data
results = labelData(testing, model, 256, .125); % label testing data
```

The third (sampling rate) and fourth (slide width) arguments of labelData are 256 Hz and 0.01 s by default. The results structure contains the following fields:

`results.label`	cell array of predicted labels for continuous data
`results.time`	two-dimensional vector with [startTime, endTime] in seconds

`results.certainty`	vector of probability-like quality indicators
`results.likelihoods`	array of probability estimates for each possible label

The code above generates the following output:

```
results =

1x3837 struct array with fields:
    label
    time
    certainty
    likelihoods
```

We can call an entry in this array using `results(1)`:

```
results(1)

ans =

          label: 'None'
           time: [0.1836 0.3047]
      certainty: 0.8704
    likelihoods: {7x1 cell}
```

There were 3837 number of slides, each slide at a width of 0.125 s long. A certainty thresholding policy can be applied to this output to reduce false positives in the data. For example:

```
results1 = thresholdPolicy(results, 'None', .5);
```

filters the relabels the data as (`'None'`) if the label certainty is less than 0.5 and one of the top two possible classes is the baseline class (`'None'`). The input arguments are:

`results`	the output from labelData
`baseline_class`	the class which is considered the baseline class used in building the model (here, ‘None’)
`certainty_threshold`	the threshold value (here, .5).

---

7. EXAMPLE 3: Manually Labeling Data

DETECT provides a graphic user interface (GUI) for manually labeling continuous time series based on EEGLAB’s `eegplot` function. This function, `getLabels`, allows you to view your data in a continuous scrolling window and to easily mark and categorize intervals in a continuous time series. This function is useful for creating training sets for your own data. This function can handle both EEGLAB EEG datasets as well as MATLAB matrix inputs where the dimensions of the matrix are (*channels* × *frames*).

A second function, `plotLabeledData`, displays labeled data based on the results of the classification. This function is useful for accurately labeling features such as artifacts and can be used as a preliminary step to manual removal of artifacts.

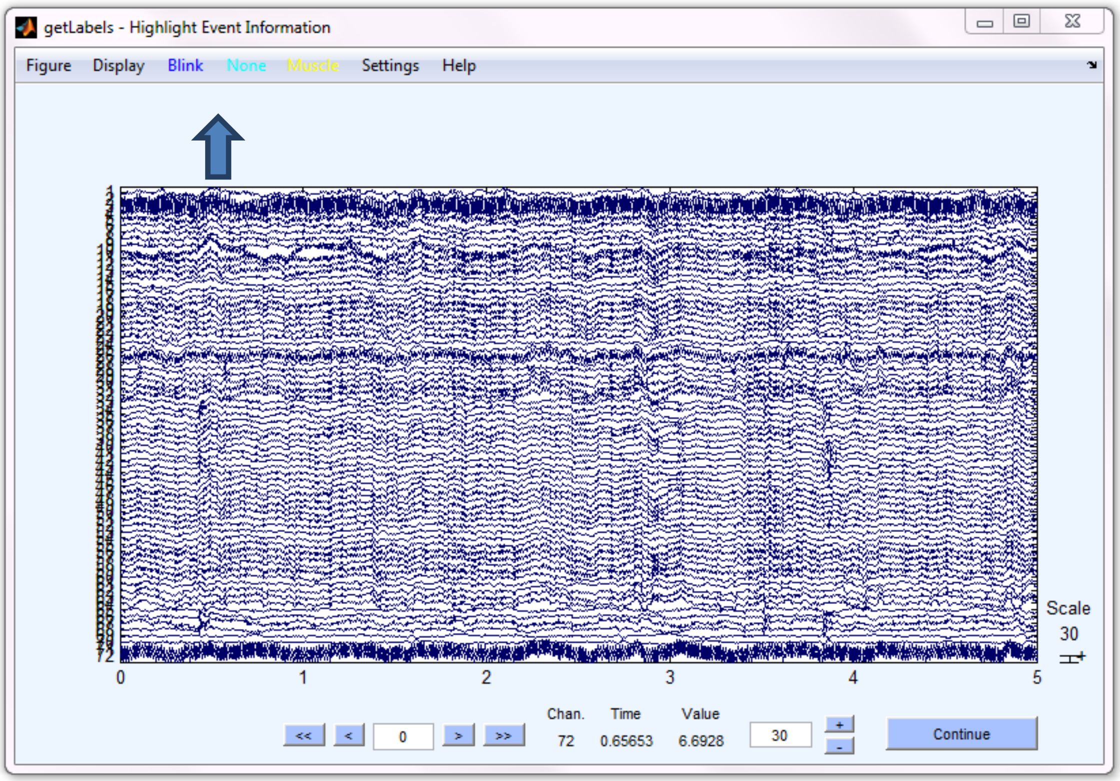
First off, load a continuous dataset:

```
load data/testing;              % load data to be manually labeled
```

You can use this command to highlight the data containing blinks and muscle artifacts, with a desired event interval length of 0.5 s:

```
[dataWindows, labels] = getLabels(testing, {'Blink', 'None', 'Muscle'}, .5)
```

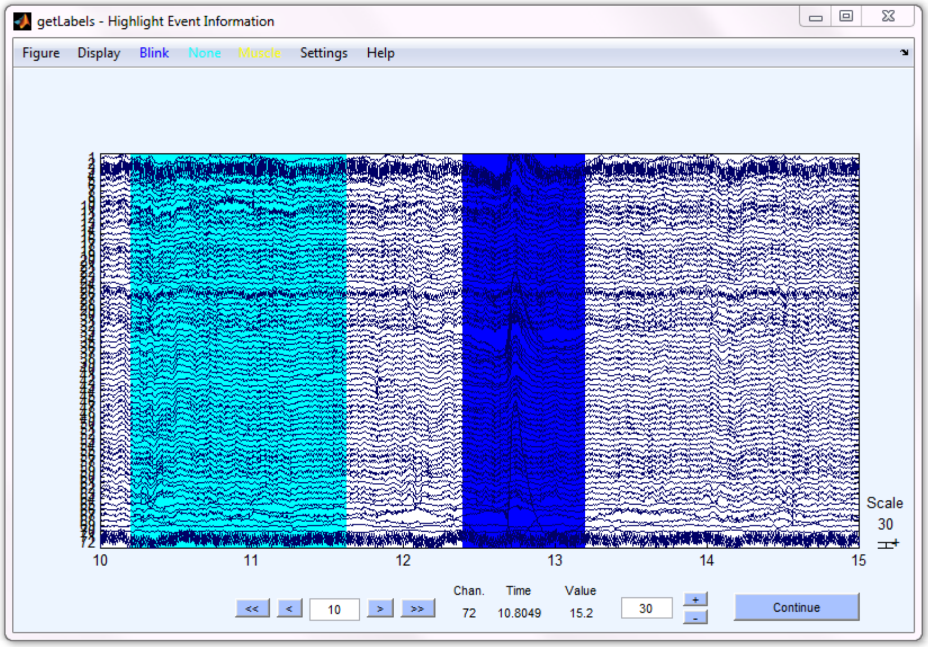
This command brings up the following GUI:



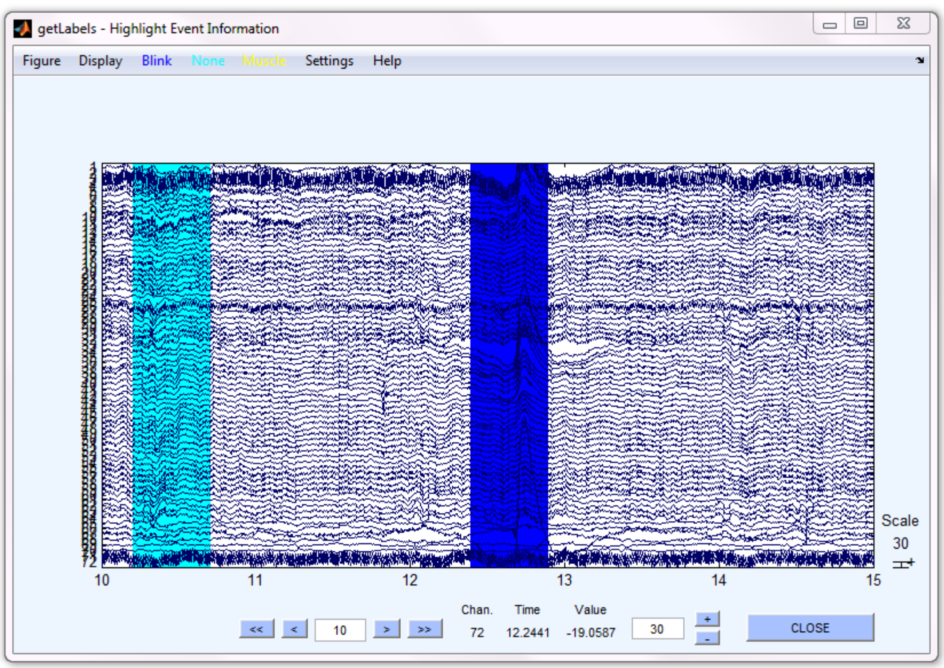
The three colored buttons on the top toolbar (“Blink”, “None”, and “Muscle”) correspond to the labels passed as the second argument to `getLabels`. These buttons determine the colors and labels of the highlighted regions. Note that if the input data was an EEGLAB EEG structure, the channel locations (vertical axis) as well as the event information will be plotted in addition to the data. Use this GUI to highlight regions of data to be labeled as follows:

1. Press the button corresponding to the label you choose.
2. Click the cursor on the data at the starting point and drag the mouse while holding the mouse button down.
3. Release the button when you have reached the end of the segment you wish to label.

It is generally a good rule of thumb to have an equal number of trials for each event. In this example, one ‘None’ region for every ‘Blink’ and ‘Muscle’ region selected helps to ensure the accurate estimation of classification models. An example of some highlighted regions is shown below.



Once you are finished highlighting your data, hit the “Continue” button at the bottom right. Another GUI will pop up, this time with all the events aligned to be exactly the length specified in the function call. The GUI that it shows for our particular example is shown below:



In the MATLAB command window, another prompt will appear:

```
Adjusting event timings for the desired event length of 0.500 seconds


Do you want to:
  1. save this labeling(s),
  2. continue labeling(c), or
  3. quit without saving(q)? [s/c/q]:
```

Here, three options are available. The first option, “s”, extracts the highlighted regions and provides a label for each region. The second option “c” is used whenever you need to adjust the highlighted regions (for example, it may not cover the desired area, or you may need to add additional events). The final option “q” will quit the function without saving the results.

Hit “Close” to close the figure after you have inspected the highlighted regions for accuracy. Typing “s” will calculate two variables, `dataWindows` and `labels`. The `dataWindows` output is an array that contains the data in windowed form, as a 3D array of size *channels* × *windowSize* × *windows*, where *windowSize* is the number of samples in a trial, and *windows* is the total number of labeled trials in the data. The `labels` output is a cell array of length *windows* that contains a label for each labeled trial. In the above example, we only highlighted two regions, so there are only two entries in the `labels` variable: `{'None'; 'Blink'}.` If the input to `getLabels` was an EEGLAB EEG structure, the output will also be an EEGLAB EEG structure whose data is a 3-D array, while if the input was a 2-D data matrix, the output will be a 3-D data matrix.

We can use the output of `getLabels` to train a model:

```
model = getModel(dataWindows, labels); % create classifier using defaults
```

(See example 1 for details on the defaults and additional parameters for `getModel`.)

---

8. EXAMPLE 4: Plotting Labeled Data

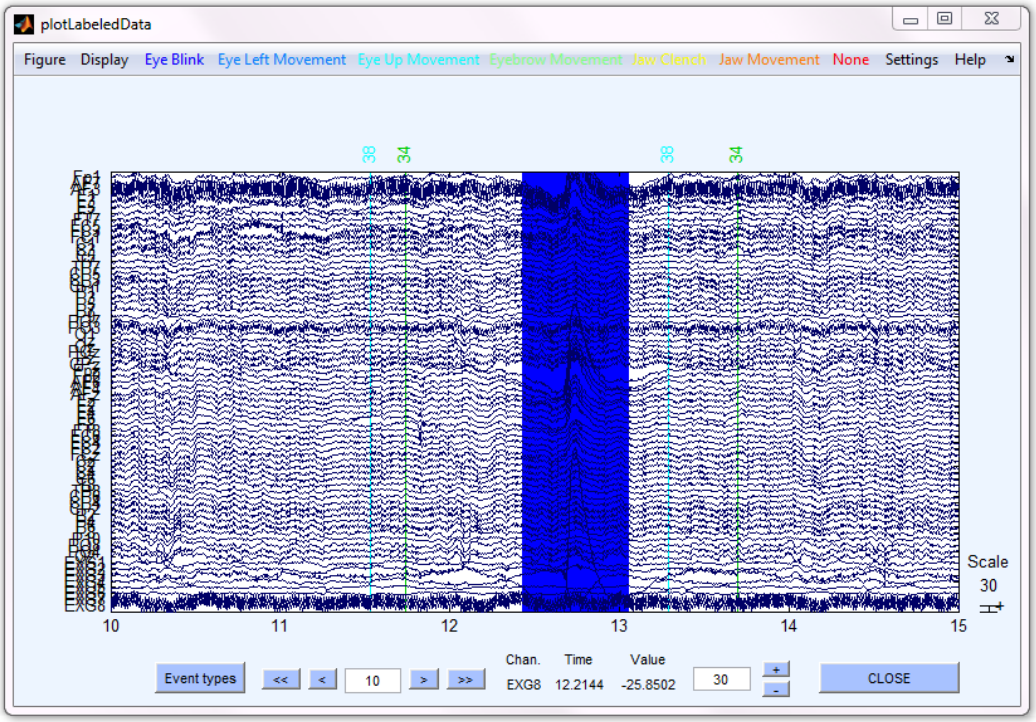
We can use DETECT to plot detected events in time series as colored segments in a scrolling plot. As an example, suppose we are interesting in labeling sections of EEG data containing artifacts. To build the artifact classification model using EEGLAB .set files:

```
training = pop_loadset('data/training.set');  % load the training data
load('data/labels.mat');                     % load labels for training data
model = getModel(training, labels); % create classifier using defaults
testing = pop_loadset('data/testing.set');  % load continuous testing data
results = labelData(testing, model, 256, .125); % label testing data
```

This code builds a continuous detection model using a slide width of 0.125 s for EEG data sampled at 256 Hz. The artifact types included in the labels are “Jaw Clench”, “Jaw Movement”, “Eye Blink”, “Eye Left Movement”, “Eye Up Movement”, and “Eyebrow Movement”.

Once the calculation finishes, we run the following code to see the effect of labeling. In the example, we are only interested in eye blinks (the fifth input argument in the function):

```
% plot the data
labelSet = plotLabeledData(testing, model, results, 'srate', ...
                           256, 'includeClasses', {'Eye Blink'})
```



All the artifact types appear as uniquely colored buttons at the top toolbar. The artifact types are "Jaw Clench", "Jaw Movement", "Eye Blink", "Eye Left Movement", "Eye Up Movement", "Eyebrow Movement" and "None". Only Eye Blinks are displayed, however.

When you push the CLOSE button, the GUI closes and returns the `labelSet` in a structure similar to this:

```
labelSet =

    'Eye Blink'    [ 12.4336]    [ 13.0547]
    'Eye Blink'    [ 19.8086]    [ 20.3047]
    'Eye Blink'    [ 25.8086]    [ 26.4297]
    'Eye Blink'    [ 40.9336]    [ 41.5547]
    'Eye Blink'    [ 93.8086]    [ 94.1797]
    'Eye Blink'    [119.0586]    [119.6797]
    'Eye Blink'    [124.1836]    [124.4297]
    'Eye Blink'    [173.4336]    [174.0547]
    'Eye Blink'    [175.1836]    [175.9297]
    'Eye Blink'    [179.0586]    [179.5547]
    'Eye Blink'    [180.4336]    [180.5547]
    'Eye Blink'    [184.5586]    [185.1797]
    'Eye Blink'    [274.1836]    [274.6797]
```

```
'Eye Blink'    [285.1836]    [285.6797]
'Eye Blink'    [296.9336]    [297.4297]
'Eye Blink'    [350.0586]    [350.6797]
'Eye Blink'    [355.6836]    [356.3047]
'Eye Blink'    [357.0586]    [357.1797]
'Eye Blink'    [357.3086]    [357.4297]
'Eye Blink'    [361.8086]    [362.6797]
'Eye Blink'    [362.8086]    [363.3047]
'Eye Blink'    [446.6836]    [447.5547]
```

The first column is the detected event, while the second and third columns denote the start and end times, respectively, of the event. In this case, we only wanted to display one event, “Eye Blink”, and so only the times where eye blinks are present are shown. If more than one type of event is chosen for display, the output of this function will show the start and end times of each event type chronologically.

---

9. EXAMPLE 5: Updating a Previously Labeled Dataset

DETECT has functionality to update a previously labeled dataset. This previous labeling can either be from a manual labeling (using the function `markEvents`) or from an automated labeling (use `plotLabeledData` on labeling generated from `LabeledData` or `LabeledWindows`).

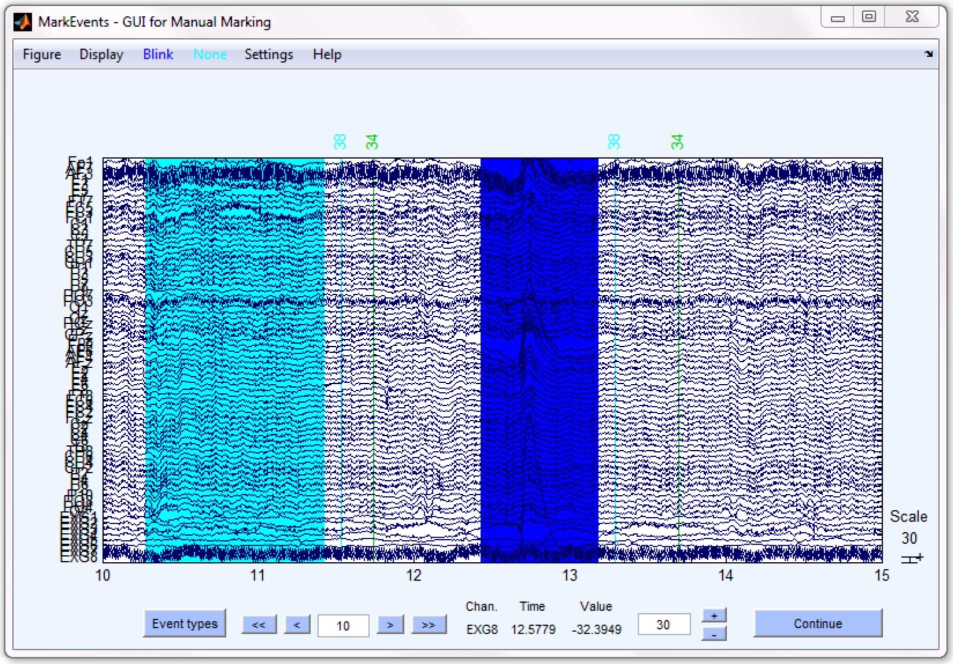
Example: Load the dataset:

```
EEG = pop_loadset('data/testing.set')
```

Then run the following command and manually highlight events with Blinks and None:

```
labelSet1 = markEvents(EEG, {'Blink', 'None'}, 'srate', 256)
```

This function call generates a GUI that is similar to `getLabels`. Use the same procedure for highlighting data as previously shown in the examples for `getLabels`. An example labeling is shown below:



Hitting Continue will output the following event structure:

```
labelSet1 =

    'None'      [10.2821]     [11.4245]     []
    'Blink'     [12.4334]     [13.1827]     []
```

The first column is the event type, the second and third columns are the start and end time in seconds, and the fourth column is an index of bad channels (here it is empty).

If you want to update the markings, use

```
labelSet2 = markEvents(EEG, {'Blink', 'Muscle'}, 'srate', 256, ...
                    'regions', labelSet1)
```

This will take the existing events field and display it on the data scroll plot. Any modifications made to the data will be automatically saved to `labelSet2` when you hit "Continue".

10. EXAMPLE 6: Comparing Two Labeled Datasets

DETECT also has functionality to automatically compare two labeled datasets. To compare two labeled datasets, first load up an EEG dataset:

```
EEG = pop_loadset('data/testing.set')
load('data/labeled-data.mat');           % load the labeled data
```

We can plot the data using `plotMarkedData`.

```
plotMarkedData(EEG, labelSet1) % plot the data
plotMarkedData(EEG, labelSet2) % plot the data
```

The first label set has two events, while the second label set has only one event.

```
labelSet1 =

    'Blink'     [0.9795]    [1.3351]    []
    'Muscle'    [1.3382]    [1.5794]    []


labelSet2 =

    'Blink'    [0.9720]    [1.5061]    []
```

We are interested in measuring the agreement between the two label sets. We allow for a timing error as an additional input in the comparison. For example, in the first label set, the 'Blink' starts a little bit later than in the second label set. The function call for this is :

```
[results errorInfo timeInfo] = compareLabels(EEG, labelSet1, labelSet2, ...
                                             0, 256)
```

The outputs look like this:

```
-----------------------------------------------------
Total Time in Agreement = 479.734 seconds
Total Time in TypeError =  0.168 seconds
Total Time in FalsePositive =  0.004 seconds
Total Time in FalseNegative =  0.070 seconds
Total Time of Data = 479.996 seconds
-----------------------------------------------------

results =

    'NullAgreement'    [     0]    [  0.9609]
    'FalsePositive'    [0.9648]    [  0.9688]
    'Agreement'        [0.9727]    [  1.3281]
    'TypeError'        [1.3320]    [  1.5000]
    'FalseNegative'    [1.5039]    [  1.5742]
    'NullAgreement'    [1.5781]    [479.9961]

errorInfo =

    'Null'      'Blink'    [0.9648]    [0.9688]
    'Muscle'    'Blink'    [1.3320]    [1.5000]
    'Muscle'    'Null'     [1.5039]    [1.5742]
```

```
timeInfo =

        agreement: 479.7344
        typeError: 0.1680
    falsePositive: 0.0039
    falseNegative: 0.0703
        totalTime: 479.9961
```

‘NullAgreement’ is the agreement between the two regions when no events are highlighted. Here, the null agreement is from time 0 to the beginning of the first event in time among the two datasets. A ‘FalsePositive’ error is generated because the ‘Blink’ in the second label set starts before the blink in the first label set. A period of ‘Agreement’ follows because both regions are labeled with the same type at the same time. Starting at time 1.3320, the first label set called the type as “Muscle”, while the second label set still called the region “Blink”. Therefore, there is a type error labeled ‘TypeError’ in the output. The ‘Muscle’ area in the first label set extends to time 1.5974, which is further than the ‘Blink’ area in label set 2, so the area generates a ‘FalseNegative’ error from 1.5039 to 1.5742. After this, there is ‘NullAgreement’ until the end of the dataset. Note that small numerical differences are expected since we convert the labeled regions into data points in frames to compare the regions.

The output `errorInfo` describes the type of errors generated from ‘FalsePositive’, ‘FalseNegative’ and ‘TypeError’ conditions. For example, the first entry in `errorInfo` describes the ‘FalsePositive’ error where Region 2 did not have a label where Region 1 did (Blink). The second region in `errorInfo` describes the TypeError where Muscle was in Region 1 while Blink was in Region 2. The third and fourth columns of this matrix denote the start and end time, in seconds, of the disagreement.

The output `timeInfo` gives a summary of the total time in each of the possible states in seconds. Here, Agreement is the total time where the two regions agree (both agree on the absence or presence of an event). The `totalTime` is the length of the data in seconds.
If the allowable timing error is 0.1 s, the output is now:

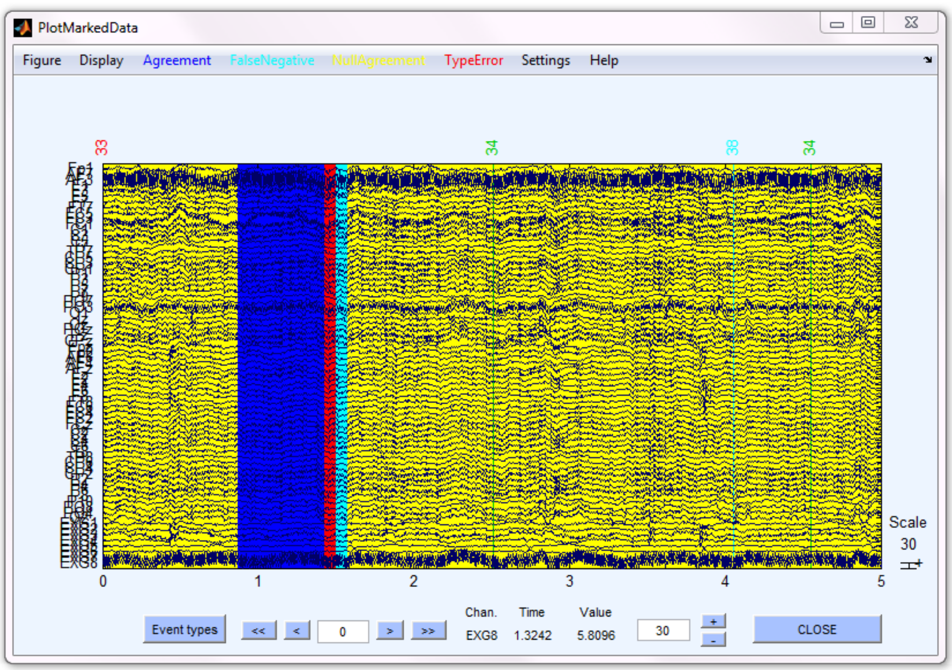
```
[results errorInfo timeInfo] = compareLabels(EEG, labelSet1, labelSet2, ...
                                             .1, 256)

-----------------------------------------------------
Total Time in Agreement = 479.840 seconds
Total Time in TypeError =  0.070 seconds
Total Time in FalsePositive =  0.000 seconds
Total Time in FalseNegative =  0.070 seconds
Total Time of Data = 479.996 seconds
-----------------------------------------------------

results =

    'NullAgreement'    [     0]    [  0.8711]
    'Agreement'        [0.8750]    [  1.4258]
    'TypeError'        [1.4297]    [  1.5000]
    'FalseNegative'    [1.5039]    [  1.5742]
    'NullAgreement'    [1.5781]    [479.9961]
```

```
errorInfo =

    'Muscle'    'Blink'    [1.4297]    [1.5000]
    'Muscle'    'Null'     [1.5039]    [1.5742]


timeInfo =

        agreement: 479.8398
        typeError: 0.0703
    falsePositive: 0
    falseNegative: 0.0703
        totalTime: 479.9961
```

The only difference here is the 'FalsePositive' is now an 'Agreement' because, while the 'Blink' in the first label set starts later than in the second label set, the start times are within 0.1 s of each other, and so the labelings are said to be equal. The agreement starts and ends 0.1 s earlier and later, respectively, to account for the allowable timing error. This also pushes the start of the 'TypeError' by 0.1 s later since the regions are concurrent. The allowable timing error is used only when the label sets have a region of the same type; no timing error adjustment is used when the label ets have different type, such as the 'FalseNegative' entry that stays the same in both scenarios.

Note that we can plot the results of `compareLabels` using the `plotMarkedData` function:

```
plotMarkedData(EEG, results, 'srate', 256)
```



CompareLabels

Compares two sets of labeled data.

Contents

Syntax

```
results = compareLabels(EEG, labeledSet1, labeledSet2, timingError, srate)
[results errorInfo] = compareLabels(EEG, labeledSet1, labeledSet2,
timingError, srate)
[results errorInfo timeInfo] = compareLabels(EEG, labeledSet1, labeledSet2,
timingError, srate)
```

Description

`results = compareLabeledData(EEG, labeledSet1, labeledSet2, timingError, srate)` returns an event structure containing the decision types, together with a start and end time, in seconds. The decision types can take one of five values:

Type name	Description
`Agreement`	The labels of the two label sets are the same and in type agreement
`TypeError`	The labels from the two label sets are the same in time but not in type agreement
`FalsePositive`	A label in label set 2 was not found in label set 1 at that time
`FalseNegative`	A label in label set 1 was not found in label set 2 at that time
`NullAgreement`	Neither label set was labeled that time

`[results, errorInfo] = compareLabeledData(EEG, labeledSet1, labeledSet2, timingError, srate)` returns an additional structure `errorInfo` which contains information about decisions with `typeError, falsePositive` and `falseNegative`.

`[results, errorInfo, timeInfo] = compareLabeledData(EEG, labeledSet1, labeledSet2, timingError, srate)` returns a summary of the time, in seconds, in each of the five states described above.

The input arguments are

Argument	Description
`inputData`	Either a 2-D matrix input or an EEGLAB EEG structure containing 2-D data. Dimensions are (channels x frames)
`labeledSet1`	The output of either `markEvents` or `plotLabeledData` (treated as ground truth)
`labeledSet2`	The output of either `markEvents` or `plotLabeledData`
`timingError`	Allowable timing error to still consider two regions as the same (in seconds) (See examples below for further details)
`srate`	Sampling rate of data in Hz

The outputs are

Argument	Description
`results`	Cell array with three columns: [agreement type], startTime], [endTime]
`errorInfo`	For events with 'TypeError', 'FalsePositive' or 'FalseNegative', will give the following output: [type1], [type2], [startTime], [endTime] (see examples below)
`timeInfo`	A structure with output fields `.agreement`, `.typeError`, `.falsePositive`, `.falseNegative`, `.totalTime`. Each field represents the total time, in seconds, of each state.

Example

Compare the labelings using two different channel sets to train an artifact discrimination model:

```
training = pop_loadset('data/training.set');
load('data/labels.mat');

% build model using all 64 EEG Channels
model1 = getModel(training, labels, 1:64);

% now build model using only 32 EEG channels
model2 = getModel(training, labels, 1:32);

% now load testing dataset
testing = pop_loadset('data/testing.set');

% Use sliding window of .125s for data sampled at 256hz
results1 = labelData(testing, model1, 256, .125);
results2 = labelData(testing, model2, 256, .125);

% apply a certainty policy to remove false positives

results1 = thresholdPolicy(results1, 'None', .5);
results2 = thresholdPolicy(results2, 'None', .5);
```

```
    % plot the data and get an event list ignoring the category 'None'

    classes = {'Eye Blink', 'Eye Left Movement', 'Eye Up Movement', 'Eyebrow
Movement', 'Jaw Clench', 'Jaw Movement'};
    labelSet1 = plotLabeledData(testing, model1, results1, 'srate', 256,
'includeClasses', classes);
    labelSet2 = plotLabeledData(testing, model2, results2, 'srate', 256,
'includeClasses', classes);

    % compare the labelings, allowing for up to .100s timing error, for
    % data sampled at 256hz.

    [results, errorInfo, timeInfo] = compareLabels(testing, labelSet1,...
    labelSet2, .1, 256)
pop_loadset(): loading file data\training.set ...
pop_loadset(): loading file data\testing.set ...

-----------------------------------------------------
Total Time in Agreement = 460.527 seconds
Total Time in TypeError =  2.219 seconds
Total Time in FalsePositive = 16.016 seconds
Total Time in FalseNegative =  0.773 seconds
Total Time of Data = 479.996 seconds
-----------------------------------------------------

results =

    'NullAgreement'    [       0]    [ 12.3281]
    'Agreement'        [ 12.3320]    [ 13.1484]
    'NullAgreement'    [ 13.1523]    [ 16.3008]
    'FalseNegative'    [ 16.3047]    [ 16.3281]
    'Agreement'        [ 16.3320]    [ 16.8984]
    'FalseNegative'    [ 16.9023]    [ 16.9258]
    'NullAgreement'    [ 16.9297]    [ 19.7031]
    'Agreement'        [ 19.7070]    [ 20.2734]
    'TypeError'        [ 20.2773]    [ 20.3008]
    'NullAgreement'    [ 20.3047]    [ 25.7031]
    'Agreement'        [ 25.7070]    [ 26.3984]
    'TypeError'        [ 26.4023]    [ 26.4258]
    'NullAgreement'    [ 26.4297]    [ 37.8008]
    'FalsePositive'    [ 37.8047]    [ 37.9258]
    'NullAgreement'    [ 37.9297]    [ 40.8281]
    'Agreement'        [ 40.8320]    [ 41.6484]
    'NullAgreement'    [ 41.6523]    [ 70.3008]
    'FalsePositive'    [ 70.3047]    [ 70.4258]
    'NullAgreement'    [ 70.4297]    [ 73.6758]
    'FalsePositive'    [ 73.6797]    [ 73.8008]
    'NullAgreement'    [ 73.8047]    [ 83.5508]
    'FalsePositive'    [ 83.5547]    [ 83.8008]
    'NullAgreement'    [ 83.8047]    [ 93.1758]
    'FalsePositive'    [ 93.1797]    [ 93.5508]
    'NullAgreement'    [ 93.5547]    [ 93.8008]
    'TypeError'        [ 93.8047]    [ 94.0508]
    'NullAgreement'    [ 94.0547]    [113.8008]
    'FalsePositive'    [113.8047]    [113.9258]
    'NullAgreement'    [113.9297]    [118.9531]
```

'Agreement'	[118.9570]	[119.6484]
'NullAgreement'	[119.6523]	[124.1758]
'FalseNegative'	[124.1797]	[124.4258]
'NullAgreement'	[124.4297]	[173.3281]
'Agreement'	[173.3320]	[173.7734]
'TypeError'	[173.7773]	[174.0508]
'NullAgreement'	[174.0547]	[175.1758]
'TypeError'	[175.1797]	[175.2031]
'Agreement'	[175.2070]	[175.6484]
'TypeError'	[175.6523]	[175.9258]
'FalsePositive'	[175.9297]	[177.3008]
'NullAgreement'	[177.3047]	[177.8008]
'FalsePositive'	[177.8047]	[178.5508]
'TypeError'	[178.5547]	[178.6758]
'FalseNegative'	[178.6797]	[178.8008]
'NullAgreement'	[178.8047]	[178.9258]
'FalseNegative'	[178.9297]	[178.9531]
'Agreement'	[178.9570]	[179.2734]
'TypeError'	[179.2773]	[179.4531]
'Agreement'	[179.4570]	[182.0234]
'TypeError'	[182.0273]	[182.0508]
'FalseNegative'	[182.0547]	[182.1758]
'NullAgreement'	[182.1797]	[184.4531]
'Agreement'	[184.4570]	[185.1484]
'TypeError'	[185.1523]	[185.1758]
'NullAgreement'	[185.1797]	[199.4258]
'FalsePositive'	[199.4297]	[199.6758]
'NullAgreement'	[199.6797]	[206.6758]
'FalsePositive'	[206.6797]	[207.4258]
'NullAgreement'	[207.4297]	[217.8008]
'FalsePositive'	[217.8047]	[218.3008]
'NullAgreement'	[218.3047]	[218.5508]
'FalsePositive'	[218.5547]	[218.9258]
'NullAgreement'	[218.9297]	[235.8008]
'FalsePositive'	[235.8047]	[236.4258]
'NullAgreement'	[236.4297]	[242.4258]
'FalsePositive'	[242.4297]	[243.0508]
'NullAgreement'	[243.0547]	[247.6758]
'FalsePositive'	[247.6797]	[248.6758]
'NullAgreement'	[248.6797]	[258.9258]
'FalsePositive'	[258.9297]	[259.5508]
'NullAgreement'	[259.5547]	[263.6758]
'FalsePositive'	[263.6797]	[264.0508]
'NullAgreement'	[264.0547]	[265.1758]
'FalsePositive'	[265.1797]	[265.9258]
'NullAgreement'	[265.9297]	[274.0781]
'Agreement'	[274.0820]	[274.7734]
'FalseNegative'	[274.7773]	[274.8008]
'NullAgreement'	[274.8047]	[285.0781]
'Agreement'	[285.0820]	[285.7734]
'NullAgreement'	[285.7773]	[292.5508]
'FalseNegative'	[292.5547]	[292.6758]
'NullAgreement'	[292.6797]	[296.9258]
'TypeError'	[296.9297]	[296.9531]
'Agreement'	[296.9570]	[297.5234]
'NullAgreement'	[297.5273]	[349.9531]
'Agreement'	[349.9570]	[350.6484]

```
    'FalseNegative'    [350.6523]    [350.6758]
    'NullAgreement'    [350.6797]    [355.5508]
    'FalseNegative'    [355.5547]    [355.5781]
    'Agreement'        [355.5820]    [356.1484]
    'TypeError'        [356.1523]    [356.1758]
    'FalsePositive'    [356.1797]    [357.0508]
    'TypeError'        [357.0547]    [357.1758]
    'FalsePositive'    [357.1797]    [357.3008]
    'TypeError'        [357.3047]    [357.4258]
    'FalsePositive'    [357.4297]    [361.0508]
    'NullAgreement'    [361.0547]    [361.5508]
    'FalsePositive'    [361.5547]    [361.9258]
    'TypeError'        [361.9297]    [362.3281]
    'Agreement'        [362.3320]    [363.1484]
    'TypeError'        [363.1523]    [363.2031]
    'Agreement'        [363.2070]    [364.0234]
    'TypeError'        [364.0273]    [364.3008]
    'NullAgreement'    [364.3047]    [365.8008]
    'FalsePositive'    [365.8047]    [366.3008]
    'NullAgreement'    [366.3047]    [382.3008]
    'FalsePositive'    [382.3047]    [382.4258]
    'NullAgreement'    [382.4297]    [446.7031]
    'Agreement'        [446.7070]    [447.5234]
    'FalseNegative'    [447.5273]    [447.5508]
    'NullAgreement'    [447.5547]    [458.8008]
    'FalsePositive'    [458.8047]    [459.3008]
    'NullAgreement'    [459.3047]    [459.4258]
    'FalsePositive'    [459.4297]    [459.8008]
    'NullAgreement'    [459.8047]    [462.8008]
    'FalsePositive'    [462.8047]    [462.9258]
    'NullAgreement'    [462.9297]    [472.3008]
    'FalsePositive'    [472.3047]    [472.6758]
    'NullAgreement'    [472.6797]    [479.9961]


errorInfo =

    'Eye Left Movement'    'Null'                 [ 16.3047]    [ 16.3281]
    'Eye Left Movement'    'Null'                 [ 16.9023]    [ 16.9258]
    'Eye Blink'            'Eye Up Movement'      [ 20.2773]    [ 20.3008]
    'Eye Blink'            'Eye Up Movement'      [ 26.4023]    [ 26.4258]
    'Null'                 'Jaw Clench'           [ 37.8047]    [ 37.9258]
    'Null'                 'Jaw Clench'           [ 70.3047]    [ 70.4258]
    'Null'                 'Jaw Clench'           [ 73.6797]    [ 73.8008]
    'Null'                 'Jaw Clench'           [ 83.5547]    [ 83.8008]
    'Null'                 'Jaw Clench'           [ 93.1797]    [ 93.5508]
    'Eye Blink'            'Jaw Clench'           [ 93.8047]    [ 94.0508]
    'Null'                 'Jaw Clench'           [113.8047]    [113.9258]
    'Eye Blink'            'Null'                 [124.1797]    [124.4258]
    'Eye Blink'            'Jaw Clench'           [173.7773]    [174.0508]
    'Eye Blink'            'Eye Up Movement'      [175.1797]    [175.2031]
    'Eye Blink'            'Jaw Clench'           [175.6523]    [175.9258]
    'Null'                 'Jaw Clench'           [175.9297]    [177.3008]
    'Null'                 'Jaw Clench'           [177.8047]    [178.5508]
    'Eye Up Movement'      'Jaw Clench'           [178.5547]    [178.6758]
    'Eye Up Movement'      'Null'                 [178.6797]    [178.8008]
    'Eye Left Movement'    'Null'                 [178.9297]    [178.9531]
```

```
    'Eye Blink'              'Jaw Clench'             [179.2773]    [179.4531]
    'Jaw Movement'           'Jaw Clench'             [182.0273]    [182.0508]
    'Eye Left Movement'      'Null'                   [182.0547]    [182.1758]
    'Eye Blink'              'Eye Up Movement'        [185.1523]    [185.1758]
    'Null'                   'Jaw Clench'             [199.4297]    [199.6758]
    'Null'                   'Jaw Clench'             [206.6797]    [207.4258]
    'Null'                   'Jaw Clench'             [217.8047]    [218.3008]
    'Null'                   'Jaw Clench'             [218.5547]    [218.9258]
    'Null'                   'Jaw Clench'             [235.8047]    [236.4258]
    'Null'                   'Jaw Clench'             [242.4297]    [243.0508]
    'Null'                   'Jaw Clench'             [247.6797]    [248.6758]
    'Null'                   'Jaw Clench'             [258.9297]    [259.5508]
    'Null'                   'Jaw Clench'             [263.6797]    [264.0508]
    'Null'                   'Jaw Clench'             [265.1797]    [265.9258]
    'Eye Blink'              'Null'                   [274.7773]    [274.8008]
    'Eye Up Movement'        'Null'                   [292.5547]    [292.6758]
    'Eye Blink'              'Eye Up Movement'        [296.9297]    [296.9531]
    'Eye Blink'              'Null'                   [350.6523]    [350.6758]
    'Eye Up Movement'        'Null'                   [355.5547]    [355.5781]
    'Eye Blink'              'Jaw Clench'             [356.1523]    [356.1758]
    'Null'                   'Jaw Clench'             [356.1797]    [357.0508]
    'Eye Blink'              'Jaw Clench'             [357.0547]    [357.1758]
    'Null'                   'Jaw Clench'             [357.1797]    [357.3008]
    'Eye Blink'              'Jaw Clench'             [357.3047]    [357.4258]
    'Null'                   'Jaw Clench'             [357.4297]    [361.0508]
    'Null'                   'Jaw Clench'             [361.5547]    [361.9258]
    'Eye Blink'              'Jaw Clench'             [361.9297]    [362.3281]
    'Eye Blink'              'Eye Left Movement'      [363.1523]    [363.2031]
    'Eye Left Movement'      'Eye Up Movement'        [363.2070]    [364.0234]
    'Eye Left Movement'      'Eyebrow Movement'       [364.0273]    [364.3008]
    'Null'                   'Jaw Clench'             [365.8047]    [366.3008]
    'Null'                   'Jaw Clench'             [382.3047]    [382.4258]
    'Eye Blink'              'Null'                   [447.5273]    [447.5508]
    'Null'                   'Jaw Clench'             [458.8047]    [459.3008]
    'Null'                   'Jaw Clench'             [459.4297]    [459.8008]
    'Null'                   'Jaw Clench'             [462.8047]    [462.9258]
    'Null'                   'Jaw Clench'             [472.3047]    [472.6758]


timeInfo =

        agreement: 460.5273
        typeError: 2.2188
    falsePositive: 16.0156
    falseNegative: 0.7734
        totalTime: 479.9961
```

getARfeatures

Estimate autoregressive feature vectors for data.

Contents

Syntax

```
featureVec = getARfeatures(data, modelOrder)
featureVec = getARfeatures(data, modelOrder, algorithm)
```

Description

`featureVec = getARfeatures(data, modelOrder)` calculates the autoregressive model coefficients of individual time windows for each channel in `data`. If `data` is of size channels x windowSize x windows, the function computes windows features each of size channels times `modelOrder`. The function returns an array with feature vectors in the rows for input into LIBSVM.

`featureVec = getARfeatures(..., algorithm)` specifies which algorithm to use for calculating the AR features. When `algorithm` is 1 (the default), the `getARfeatures` function uses the `arburg` function which is part of the MATLAB signal processing toolbox. If `algorithm` is 2, `getARfeatures` uses `arfit2` which is part of the TSA toolbox included with this toolbox.

Example

Create AR feature vectors using order two AR models for random data with 64 channels:

```
data = random('normal', 0, 1, [64, 1000, 10]);
featureVec = getARfeatures(data, 2);
```

Notes

The `arburg.m` function is part of the MATLAB signal processing toolbox. If you don't have that toolbox, use `arfit2.m`, which is part of TSA (Time Series Analysis) toolbox, which is distributed with this package.

getLabels

creates a windowed dataset from highlighted data segments

Contents

Syntax

```
 [dataWindows, labels] = getLabels(inputData, categories, windowLength)
 [dataWindows, labels] = getLabels(inputData, categories, windowLength,
'param1', value1, ...)
```

Description

`[dataWindows, labels] = getLabels(inputData, categories, windowLength)` opens a GUI that allows the user to select regions of the dataset with the events found in `categories`, and returns a windowed (epoched) dataset and a labels vector that can be used with getModel to train a classification model.

`[dataWindows, labels] = getLabels(..., 'param1', value1,...)` specifies additional parameters to be used.

The required input arguments are:

Argument	Description
`inputData`	An EEGLAB EEG structure containing continuous 2D EEG data or a 2D matrix array of size (channels x frames)
`categories`	A cell array of strings specifying the categories used to tag the data. Each category value will have its own button on the toolbar for easy highlighting of events.
`windowLength`	The length of a window in seconds for training (see notes).

The optional inputs are passed in as name-value pairs:

Name	Description
`'srate'`	Sampling rate of the data.
`'events'`	An array of structures with a .type and .latency field. Both fields are numeric. The field .latency is represented in frames.

'chanlocs'	An array of structures with a .labels field which is a string label denoting the channel name
'colors'	A color matrix of size (categories x 3) used to set the category buttons to specific colors.

The output arguments are:

Argument	Description
dataWindows	Either an EEG structure with windowed data, or a 3D matrix. If the original input data was an EEG structure, the output will be an EEG structure. If the input data is a 2D matrix, a 3D matrix of size (channels x windowSize x windows) is returned. The length of a window is windowLength * srate.
labels	A cell array of strings containing the label identifier for each trial.

Notes

While `getLabels` allows you to highlight regions of any size, it will re-align the highlighted sections so that they are exactly the size of windowLength from the user input. The features that we are extracting all assume that the length of data is the same for every condition.

The EEGLAB EEG structure may be passed as `inputData`.

Example

Extract 1/2 second training epochs labeled 'None' and 'Blink' using an EEGLAB EEG dataset.

```
EEG = pop_loadset('data/testing.set');
[dataWindows, labels] = getLabels(EEG, {'None', 'Blink'}, 0.5, 'srate', 256)
```

This example works the same with a 2-D matrix as input:

```
EEG = load('data/testing.mat');
[dataWindows, labels] = getLabels(testing, {'None', 'Blink'}, .5, 'srate',
256)
```

getModel

calculate an SVM model for classification

Contents

Syntax

```
 model = getModel(training, labels)
 model = getModel(training, labels, sChannels)
 model = getModel(training, labels, sChannels, numCVs)
 model = getModel(training, labels, sChannels, numCVs, featureFunction,
varargin)
```

Description

`model = getModel(training, labels)` returns a model structure containing the fitted model for classifying the input `training` into the classes of `labels`. By default, `getModel` uses all the channels in the data and 4 cross validations. The default feature function uses the autoregressive coefficients of model order two, computed for each channel and concatenated across all the channels.

`model = getModel(training, labels, sChannels)` builds the classification model using a channel index specified by `sChannels`. `sChannels` is a numeric vector of channel indices (for example, `sChannels = 1:32` specifies the first 32 channels in the data will be used).

`model = getModel(training, labels, sChannels, numCVs)` will change the number of cross-validations to use.

`model = getModel(training, labels, sChannels, numCVs, featureFunction, varargin)` builds the classification model using the feature extraction function `featureFunction`, together with its required inputs `varargin`.

Notes

The input arguments to `getModel` are:

Argument	Description
`training`	Either a 3D matrix of size channels x windowSize x windows, or an EEGLAB EEG data structure containing epoched data.
`labels`	A cell array of strings of length windows to denote a class label for each window
`sChannels`	A numeric vector of channels to use in the model building. Default is to use all available channels
`numCVs`	A numeric value to denote the number of cross validations to use
`featureFunction, varargin`	The feature function to use in the model training. The inputs needed for the feature function are passed by varargin. See getARfeatures.m for an example of a feature extraction function.

The output arguments are:

Argument	Description
`.SVM`	The SVM model structure obtained from LibSVM
`.CV`	Cross-validation accuracy
`.bestc, .bestg`	Optimal parameters for the SVM based on using a grid-search
`.alphaLabelOrder`	The alphabetical order of the labels
`.SVMLabelOrder`	Original order of label appearance in data
`.tframes`	Size of the training windows, in frames
`.sChannels`	Channel index used for training
`.ffunc, .ffunc_inputs`	Feature function used together with the inputs

Example

Build a classification model using only the first 32 channels in the dataset, using an order 5 autogressive model as features. Use 2 cross validations as well. Use the sample training dataset provided with the toolbox for illustration.

```
    load data/training.mat;
    load data/labels.mat;
    model = getModel(training, labels, 1:32, 2, @getARfeatures, 5)
model =

                SVM: [1x1 struct]
                 CV: 97.8571
              bestc: 32
```

```
              bestg: 0.0442
    alphaLabelOrder: {7x1 cell}
      SVMLabelOrder: {7x1 cell}
            tframes: 128
          sChannels: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32]
              ffunc: @getARfeatures
       ffunc_inputs: {[5]  [1]}
```

See also

getARfeatures

labelData

Create a structure containing labels with certainty measure for data

Contents

Syntax:

```
results = labelData(inputData, model)
results = labelData(inputData, model, srate)
results = labelData(inputData, model, srate, slideWidth)
```

Description

`results = labelData(inputData, model)` returns a structure containing the classification results for `inputData` using model (as computed from `getModel`). A default sampling rate of 256 Hz and a default window slide width of 0.01 seconds are used.

`results = labelData(..., srate)` uses a sampling rate of `srate` Hz for the calculation.

`results = labelData(..., slideWidth)` uses a window slide width of `slideWidth` seconds in performing the labeling.

Notes

The output structure `results` has the following fields:

Field	Description	Sample value
`.label`	Predicted label	`'None'`
`.time`	Time in seconds of the predicted label	`[10.6836 10.8047]`
`.certainty`	Measure indicating likelihood that prediction is correct	`0.925`
`.likelihoods`	Cell array of labels ordered from most likely to least likely for that event	`{7x1 cell}`

Example

Create a model from the training data and label continuous data using a sampling rate of 256 Hz and a sliding window of 250 ms.

```
   load training.mat;
   load labels.mat;
   load testing.mat;
   model = getModel(training, labels);
   results = labelData(testing, model, 256, 0.25)
results =

1x1919 struct array with fields:
    label
    time
    certainty
    likelihoods
```

Extended Notes

The certainty is calculated by using `'-b 1'` option in `LibSVM` to return the probabilities of the possible labels for each window. The `labelData` function calculates the `certainty` as (P(1)-P(2))/P(1), where P(1) is the probability of the most probable label and P(2) is the probability of the second most probable label.

labelWindows

Classify data windows using an SVM model and compare to original labels

Contents

Syntax:

```
results = labelWindows(inputData, model)
[results accuracy] = labelWindows(inputData, model, actualLabels)
```

Description

`results = labelWindows(inputData, model)` returns an array of structures containing the classification results of `inputData` based on `model`.

`[results, accuracy] = labelWindows(inputData, model, actualLabels)` returns the classification accuracy in the field `accuracy`. The `actualLabels` must be passed in order to compute the accuracy.

The input arguments are:

Argument	Description
`inputData`	Either a 3-dimensional matrix of size channels x windowSize x windows, or an EEGLAB EEG data structure containing epoched data.
`model`	The output model structure from getModel.
`actualLabels`	(Optional) A cell array of strings denoting the true class labels for `inputData.` Must be of length windows.

The output argument is an array of structures with the following fields:

Field	Description
`.label`	String label with the classified class
`.actualLabel`	The original label for the window. This will be empty if the input actualLabels was omitted.
`.certainty`	The certainty of the prediction

`.likelihoods`	The order of the categories, from most likely to least likely. The first entry of `.likelihoods` is the same as `.label`.
`.prob_estimates`	The estimated probability distribution of each category

Example

Build a classification model using only the first 32 channels in the dataset, using an order 5 autogressive model as features. Use 2 cross validations as well. Use the sample training dataset provided with the toolbox for illustration. Use the output to classify the same data.

```
    load data/training.mat;
    load data/labels.mat;
    model = getModel(training, labels, 1:32, 2, @getARfeatures, 5)
    [results accuracy] = labelWindows(training, model, labels)
    results(10)
```

```
model =

                SVM: [1x1 struct]
                 CV: 97.8571
              bestc: 32
              bestg: 0.0442
    alphaLabelOrder: {7x1 cell}
      SVMLabelOrder: {7x1 cell}
            tframes: 128
          sChannels: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
22 23 24 25 26 27 28 29 30 31 32]
              ffunc: @getARfeatures
       ffunc_inputs: {[5]  [1]}


results =

1x140 struct array with fields:
    label
    actualLabel
    certainty
    likelihoods
    prob_estimates
    labelOrder


accuracy =

   100


ans =

             label: 'None'
       actualLabel: 'None'
         certainty: 0.9406
       likelihoods: {7x1 cell}
    prob_estimates: [0.8449 0.0158 0.0117 0.0138 0.0431 0.0502 0.0206]
        labelOrder: {7x1 cell}
```

markEvents

Marks data in EEG Dataset and returns structure of eventList

Contents

Syntax

```
labelSet = markEvents(inputData, categories)
labelSet = markEvents(inputData, categories, 'param1', value1, ...)
```

Description

`labelSet = markEvents(inputData, categories)` opens a GUI that can be used for manual labeling of inputData using the categories found in `categories`.

`labelSet = markEvents(..., 'param1', value1, ...)` specifies additional parameters to be used.

The required input arguments are:

Argument	Description
`inputData`	Either a 2D matrix of dimensions channels x frames or an EEGLAB EEG data file containing 2-D data.
`categories`	Cell array of strings to label data with

The optional inputs are passed as name-value pairs:

Name	Description
`'srate'`	Sampling rate of the data
`'regions'`	Previous output of markEvents
`'chanlocs'`	An array of structures with a .labels field which is a string label denoting the channel name
`'event'`	An array of structures with a `.type and .latency field. Both fields are numeric. The field .latency is represented in frames.`

The output arguments are:

Argument	Description
`labelSet`	A matrix of cells with entries: [category], [startTime], [endTime], [badChnList]. Category is a string, startTime and endTime are numeric entries and badChnList is a numeric vector to denote bad channels.

Example

Example 1 for marking EEG data with blinks, muscles and Other and saves it to the output variable regions with a sampling rate of 256Hz.

```
   load data/testing;
   regions = markEvents(testing, {'Blink', 'Muscle', 'Other'}, 'srate', 256);
```

A sample output is:

```
 regions =
   'Blink'     [1.4542]    [2.7005]    []
   'Muscle'    [3.6723]    [4.5773]    []
```

Click on any of the toolbar buttons to label continuous data. After labeling some data, if you want to redo the markings:

```
   new_regions = markEvents(testing, {'Blink', 'Muscle', 'Other'}, 'srate',
256, 'regions', regions);
```

plotLabeledData

plots the labeled data from labelData

Contents

- Syntax
- Description
- Example

Syntax

```
labelSet = plotLabeledData(inputData, model, results)
labelSet = plotLabeledData(inputData, model, results, 'param1', value1, ...)
```

Description

`labelSet = plotLabeledData(inputData, model, results)` displays a data scroll plot window with the event information taken from `labelData`.

`labelSet = plotLabeledData(..., 'param1', value1, ...)` specifies additional parameters to be used.

The required input arguments are:

Arguments	Description
`inputData`	An EEGLAB EEG structure containing continuous 2D EEG data or a 2D matrix array of size (channels x frames)
`model`	The SVM model output from `getModel`.
`results`	The output from labelData

The optional inputs are passed in as name-value pairs:

Name	Description
`'srate'`	Sampling rate of the data
`'includeClasses'`	Cell array of strings denoting the desired plotting categories or labels (all by default)
`'eventList'`	An array of structures with a .type and .latency field. Both fields are numeric. The field .latency is represented in frames.
`'chanlocs'`	An array of structures with a `.labels` field which is a string label denoting the channel name.

The output argument is:

Argument	Description
`labelSet`	A cell array with columns [eventtype], [startTime] and [endTime]

Example

Build the artifact classification model from the sample data included in the toolbox, and display only eye blinks and jaw clenches:

```
   load data/training;
   load data/labels;
   model = getModel(training, labels, 1 : 64);
   load data/testing;
   results = labelData(testing, model, 256, .125);
   labelSet = plotLabeledData(testing, model, results, 'srate', 256,
'includeClasses', {'Eye Blink', 'Jaw Clench'})
labelSet =

    'Eye Blink'     [ 12.4336]    [ 13.0547]
    'Eye Blink'     [ 19.8086]    [ 20.3047]
    'Eye Blink'     [ 25.8086]    [ 26.4297]
    'Eye Blink'     [ 40.9336]    [ 41.6797]
    'Eye Blink'     [ 46.4336]    [ 46.6797]
    'Eye Blink'     [ 55.5586]    [ 55.9297]
    'Eye Blink'     [ 69.1836]    [ 69.3047]
    'Eye Blink'     [ 93.6836]    [ 94.3047]
    'Eye Blink'     [ 97.3086]    [ 97.5547]
    'Eye Blink'     [105.6836]    [105.8047]
    'Eye Blink'     [119.0586]    [119.6797]
    'Eye Blink'     [120.3086]    [120.4297]
    'Eye Blink'     [123.9336]    [124.4297]
    'Eye Blink'     [137.4336]    [137.8047]
    'Eye Blink'     [139.5586]    [139.9297]
    'Eye Blink'     [164.3086]    [164.5547]
    'Eye Blink'     [167.8086]    [168.0547]
    'Eye Blink'     [173.4336]    [174.0547]
    'Eye Blink'     [175.1836]    [176.0547]
    'Eye Blink'     [177.1836]    [177.3047]
    'Eye Blink'     [178.1836]    [178.3047]
    'Eye Blink'     [178.4336]    [178.5547]
    'Eye Blink'     [179.0586]    [179.5547]
    'Jaw Clench'    [181.5586]    [181.9297]
    'Eye Blink'     [184.5586]    [185.1797]
    'Eye Blink'     [228.1836]    [228.3047]
    'Eye Blink'     [235.6836]    [235.9297]
    'Eye Blink'     [268.1836]    [268.4297]
    'Eye Blink'     [269.8086]    [270.1797]
    'Eye Blink'     [274.1836]    [274.8047]
    'Eye Blink'     [285.1836]    [285.6797]
    'Eye Blink'     [296.9336]    [297.4297]
    'Eye Blink'     [350.0586]    [350.6797]
    'Eye Blink'     [355.6836]    [356.4297]
```

```
'Eye Blink'      [356.5586]     [357.8047]
'Eye Blink'      [358.1836]     [358.3047]
'Eye Blink'      [358.9336]     [359.3047]
'Eye Blink'      [361.8086]     [362.6797]
'Eye Blink'      [362.8086]     [363.1797]
'Eye Blink'      [405.3086]     [405.4297]
'Eye Blink'      [422.0586]     [422.1797]
'Eye Blink'      [446.6836]     [447.5547]
```

plotMarkedData

Plots the marked data from markEvents

Contents

Syntax

```
[] = plotMarkedData(inputData, regions)
[] = plotMarkedData(inputData, regions, 'param1', value1, ...)
```

Description

`[] = plotMarkedData(inputData, regions)` plots a GUI of the output of either `markEvents` or `plotLabeledData` using the data `inputData`.

`[] = plotMarkedData(inputData, regions, 'param1', value1, ...)` specifies additional parameters to be used.

The required input arguments are:

Argument	Description
`inputData`	Data of either a 3D matrix or an EEGLAB EEG structure containing 3D (windowed/epoched) data.
`regions`	Previous output of either `markEvents` or `plotLabeledData`

The optional inputs are passed in as name-value pairs:

Name	Description
`'srate'`	Sampling rate of the data
`'includeClasses'`	Cell array of strings denoting the desired plotting categories or labels (all by default)
`'eventList'`	An array of structures with a `.type` and `.latency` field. Both fields are numeric. The field `.latency` is represented in frames.
`'chanlocs'`	An array of structures with a `.labels` field which is a string label denoting the channel name.

plotWindowData

plots the decoded windows from labelWindows

Contents

- Syntax
- Description
- Example

Syntax

```
events = plotWindowData(inputData, model, results)
events = plotWindowData(inputData, model, results, 'param1', value1, ...)
```

Description

`events = plotWindowData(inputData, model, results)` plots the decoded windows (epochs) obtained from `labelWindows` in a scroll plot GUI. The inputs `model` and `results` come from `getModel` and `labelWindows`, respectively.

`events = plotWindowData(inputData, model, results, 'param1', value1, ...)` specifies additional parameters to be used.

The required input arguments are:

Arguments	Description
`inputData`	Data of either a 3D matrix or an EEGLAB EEG structure containing 3D (windowed/epoched) data.
`model`	The SVM model output from getModel.
`results`	The output from labelWindows

The optional input arguments are passed as name-value pairs:

Name	Description
`'srate'`	Sampling rate of the data
`'includeClasses'`	Cell array of strings denoting the desired plotting categories or labels (all by default)
`'eventList'`	An array of structures with a `.type` and `.latency` field. Both fields are numeric. The field `.latency` is represented in frames.

'chanlocs'	An array of structures with a .labels field which is a string label denoting the channel name.
'colors'	Optional; a nEvents x 3 array of custom-defined colors

The output argument is:

Argument	Description
events	a nWindows x 2 cell array with columns [eventtype] and [certainty]

Example

Build a training model on epoched data and test the model on the same epoched data. Plot only epochs containing eye blinks and jaw clenches.

```
   load data/training
   load data/labels
   model = getModel(training, labels, 1 : 64);
   results = labelWindows(training, model, labels);
   events = plotWindowData(training, model, results, 'srate', 256,
'includeClasses', {'Eye Blink', 'Jaw Clench'})
events =

    'None'                 [0.9591]
    'None'                 [0.9426]
    'None'                 [0.8901]
    'None'                 [0.9022]
    'None'                 [0.7537]
    'None'                 [0.9689]
    'None'                 [0.9712]
    'None'                 [0.9129]
    'None'                 [0.9315]
    'None'                 [0.9680]
    'None'                 [0.9323]
    'None'                 [0.9836]
    'None'                 [0.9556]
    'None'                 [0.9656]
    'None'                 [0.9570]
    'None'                 [0.8566]
    'None'                 [0.9607]
    'None'                 [0.9329]
    'None'                 [0.9700]
    'None'                 [0.4836]
    'Jaw Clench'           [0.9554]
    'Jaw Clench'           [0.9723]
    'Jaw Clench'           [0.9518]
    'Jaw Clench'           [0.9457]
    'Jaw Clench'           [0.9451]
    'Jaw Clench'           [0.9607]
    'Jaw Clench'           [0.9655]
    'Jaw Clench'           [0.9752]
    'Jaw Clench'           [0.9627]
    'Jaw Clench'           [0.9795]
```

```
'Jaw Clench'            [0.9048]
'Jaw Clench'            [0.9280]
'Jaw Clench'            [0.9093]
'Jaw Clench'            [0.9025]
'Jaw Clench'            [0.9067]
'Jaw Clench'            [0.8824]
'Jaw Clench'            [0.9038]
'Jaw Clench'            [0.9641]
'Jaw Clench'            [0.8308]
'Jaw Clench'            [0.7881]
'Jaw Movement'          [0.8738]
'Jaw Movement'          [0.9502]
'Jaw Movement'          [0.8596]
'Jaw Movement'          [0.9777]
'Jaw Movement'          [0.9719]
'Jaw Movement'          [0.9389]
'Jaw Movement'          [0.9571]
'Jaw Movement'          [0.9503]
'Jaw Movement'          [0.8952]
'Jaw Movement'          [0.9694]
'Jaw Movement'          [0.9501]
'Jaw Movement'          [0.9531]
'Jaw Movement'          [0.1885]
'Jaw Movement'          [0.9672]
'Jaw Movement'          [0.2931]
'Jaw Movement'          [0.9297]
'Jaw Movement'          [0.9359]
'Jaw Movement'          [0.9584]
'Jaw Movement'          [0.9380]
'Jaw Movement'          [0.9391]
'Eye Blink'             [0.8445]
'Eye Blink'             [0.9233]
'Eye Blink'             [0.9860]
'Eye Blink'             [0.9350]
'Eye Blink'             [0.9018]
'Eye Blink'             [0.9345]
'Eye Blink'             [0.9727]
'Eye Blink'             [0.9738]
'Eye Blink'             [0.9852]
'Eye Blink'             [0.9736]
'Eye Blink'             [0.5206]
'Eye Blink'             [0.9640]
'Eye Blink'             [0.9410]
'Eye Blink'             [0.9470]
'Eye Blink'             [0.9845]
'Eye Blink'             [0.9733]
'Eye Blink'             [0.9863]
'Eye Blink'             [0.9772]
'Eye Blink'             [0.9232]
'Eye Blink'             [0.9297]
'Eye Left Movement'     [0.9014]
'Eye Left Movement'     [0.9610]
'Eye Left Movement'     [0.9654]
'Eye Left Movement'     [0.9365]
'Eye Left Movement'     [0.9109]
'Eye Left Movement'     [0.7979]
'Eye Left Movement'     [0.8831]
```

```
'Eye Left Movement'    [0.9760]
'Eye Left Movement'    [0.9171]
'Eye Left Movement'    [0.9450]
'Eye Left Movement'    [0.8768]
'Eye Left Movement'    [0.9382]
'Eye Left Movement'    [0.8759]
'Eye Left Movement'    [0.9544]
'Eye Left Movement'    [0.9489]
'Eye Left Movement'    [0.9466]
'Eye Left Movement'    [0.9158]
'Eye Left Movement'    [0.9405]
'Eye Left Movement'    [0.9587]
'Eye Left Movement'    [0.9086]
'Eye Up Movement'      [0.9327]
'Eye Up Movement'      [0.7729]
'Eye Up Movement'      [0.7706]
'Eye Up Movement'      [0.6293]
'Eye Up Movement'      [0.9683]
'Eye Up Movement'      [0.2174]
'Eye Up Movement'      [0.9506]
'Eye Up Movement'      [0.9549]
'Eye Up Movement'      [0.9567]
'Eye Up Movement'      [0.9347]
'Eye Up Movement'      [0.9272]
'Eye Up Movement'      [0.9725]
'Eye Up Movement'      [0.9629]
'Eye Up Movement'      [0.9300]
'Eye Up Movement'      [0.9433]
'Eye Up Movement'      [0.9287]
'Eye Up Movement'      [0.9041]
'Eye Up Movement'      [0.9535]
'Eye Up Movement'      [0.9195]
'Eye Up Movement'      [0.9593]
'Eyebrow Movement'     [0.9131]
'Eyebrow Movement'     [0.9399]
'Eyebrow Movement'     [0.9388]
'Eyebrow Movement'     [0.9542]
'Eyebrow Movement'     [0.9405]
'Eyebrow Movement'     [0.9663]
'Eyebrow Movement'     [0.9604]
'Eyebrow Movement'     [0.9500]
'Eyebrow Movement'     [0.9530]
'Eyebrow Movement'     [0.9548]
'Eyebrow Movement'     [0.9666]
'Eyebrow Movement'     [0.9511]
'Eyebrow Movement'     [0.9441]
'Eyebrow Movement'     [0.8951]
'Eyebrow Movement'     [0.9312]
'Eyebrow Movement'     [0.9237]
'Eyebrow Movement'     [0.9387]
'Eyebrow Movement'     [0.9370]
'Eyebrow Movement'     [0.9403]
'Eyebrow Movement'     [0.9372]
```

thresholdPolicy

Relabel uncertain events as baseline under certain conditions

Contents

Syntax

```
  results = thresholdPolicy(results, baseline_class, certainty_threshold)
  [results accuracy] = thresholdPolicy(results, baseline_class,
certainty_policy)
```

Description

`results = thresholdPolicyPolicy(results, baseline_class, certainty_threshold)` applies a filter based on the certainty to event labels contained in the `label` field of the `results` structure. In particular, `thresholdPolicy` relabels an event as the `baseline_class` if the certainty of its most likely label is below the `certainty_threshold` and one of the top two most likely event labels is the `baseline_class`. The `baseline_class` should be a string that is one of original labels used in the model building step.

`[results accuracy] = thresholdPolicy(results, baseline_class, certainty_threshold)` recalculates the classification accuracy if the input was from `labelWindows`.

Example

Create a model from the training data and relabel uncertain events

```
   load training.mat;
   load labels.mat;
   load testing.mat;
   model = getModel(training, labels);
   results = labelData(testing, model, 256, 0.25);
   results = thresholdPolicy(results, 'None', 0.50);
```

Notes

The output structure `results` has the following fields:

Field	Description	Sample value
`.label`	Predicted label, given as a cell array of strings	`'None'`
`.time`	Time in seconds of the predicted label given as [start end] in seconds	`[10.6836 10.8047]`
`.certainty`	Measure indicating likelihood that prediction is correct	`0.925`
`.likelihoods`	Cell array of labels ordered from most likely to least likely for that event	`{7x1 cell}`

The `thresholdPolicy` compares the value of the `results.certainty` entry with the `certainty_threshold`. If this value is below the threshold and one of the top two most likely labels (found in the first two entries of `results.likelihoods`) is the `baseline_class`, then `thresholdPolicy` changes the label to be `baseline_class`.

This is a conservative policy because if there is any possibility that the data could be the `baseline_class`, the class is relabeled to be the `baseline_class`.

unknownPolicy

Relabel uncertain non-baseline events as "Unknown"

Contents

- Syntax
- Description
- Example
- Notes

Syntax

```
  results = unknownPolicy(results, baseline_class, certainty_threshold)
  [results accuracy] = unknownPolicy(results, baseline_class,
certainty_threshold)
```

Description

`results = unknownPolicy(results, baseline_class, certainty_threshold)` applies a filter to event labels based on the certainty. after classification to relabel events based on the certainty. In particular, `unknownPolicy` relabels an event as `"Unknown"` if the certainty of its most likely event is below the `certainty_threshold` and neither of the top two most likely event labels is the `baseline_class`. The output `results` structure is the same as the input `results` structure except that its label fields are adjusted to reflect the certainty policy.

`[results accuracy] = unknownPolicy(results, baseline_class, certainty_threshold)` recalculates the classification accuracy if the input was from `labelWindows`.

Example

Create a model from the training data and relabel uncertain events

```
   load('data/training.mat');
   load('data/labels.mat');
   load('data/testing.mat');
   model = getModel(training, labels);
   results = labelData(testing, model, 256, 0.25);
   results = unknownPolicy(results, 'None', 0.5);
```

Notes

The output is an array of structures the following fields:

Field	Description	Sample value
`.label`	Predicted label, given as a cell array of strings	`'None'`
`.time`	Time in seconds of the predicted label given as [start end] in seconds	`[10.6836 10.8047]`
`.certainty`	Measure indicating likelihood that prediction is correct	`0.925`
`.likelihoods`	Cell array of labels ordered from most likely to least likely for that event	`{7x1 cell}`

The `unknownPolicy` compares the value of the `results.certainty` entry with the `certainty_threshold`. If this value is below the threshold and one of the top two most likely labels (found in the first two entries of `results.likelihoods`) is the `baseline_class`, then `unknownPolicy` changes the label to be `baseline_class`.

The `unknownPolicy` differs from `thresholdPolicy` in that if the certainty is low and one of the top two predicted classes is not `baseline_class`, it will relabel the data to be `'Unknown'`. This is helpful for finding interesting sections of the data that do not belong confidently to any of the categories found in the original training set.

NO. OF COPIES	ORGANIZATION
1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM C A DAVISON 320 MANSCEN LOOP STE 115 FORT LEONARD WOOD MO 65473
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM D T DAVIS BLDG 5400 RM C242 REDSTONE ARSENAL AL 35898-7290
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRS EA DR V J RICE BLDG 4011 RM 217 1750 GREELEY RD FORT SAM HOUSTON TX 78234-5002
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM DG J RUBINSTEIN BLDG 333 PICATINNY ARSENAL NJ 07806-5000
1 (PDF)	ARMY RSCH LABORATORY – HRED ARMC FIELD ELEMENT RDRL HRM CH C BURNS THIRD AVE BLDG 1467B RM 336 FORT KNOX KY 40121
1 (PDF)	ARMY RSCH LABORATORY – HRED AWC FIELD ELEMENT RDRL HRM DJ D DURBIN BLDG 4506 (DCD) RM 107 FORT RUCKER AL 36362-5000
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM CK J REINHART 10125 KINGMAN RD BLDG 317 FORT BELVOIR VA 22060-5828

NO. OF COPIES	ORGANIZATION
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM AY M BARNES 2520 HEALY AVE STE 1172 BLDG 51005 FORT HUACHUCA AZ 85613-7069
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM AP D UNGVARSKY POPE HALL BLDG 470 BCBL 806 HARRISON DR FORT LEAVENWORTH KS 66027-2302
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM AT J CHEN 12423 RESEARCH PKWY ORLANDO FL 32826-3276
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM AT C KORTENHAUS 12350 RESEARCH PKWY ORLANDO FL 32826-3276
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM CU B LUTAS-SPENCER 6501 E 11 MILE RD MS 284 BLDG 200A 2ND FL RM 2104 WARREN MI 48397-5000
1 (PDF)	ARMY RSCH LABORATORY – HRED FIRES CTR OF EXCELLENCE FIELD ELEMENT RDRL HRM AF C HERNANDEZ 3040 NW AUSTIN RD RM 221 FORT SILL OK 73503-9043
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM AV W CULBERTSON 91012 STATION AVE FORT HOOD TX 76544-5073
1 (PDF)	ARMY RSCH LABORATORY – HRED HUMAN RSRCH AND ENGRNG DIRCTRT MCOE FIELD ELEMENT RDRL HRM DW C CARSTENS 6450 WAY ST BLDG 2839 RM 310 FORT BENNING GA 31905-5400
1 (PDF)	ARMY RSCH LABORATORY – HRED RDRL HRM DE A MARES 1733 PLEASONTON RD BOX 3 FORT BLISS TX 79916-6816

NO. OF COPIES	ORGANIZATION
8 (PDF)	ARMY RSCH LABORATORY – HRED SIMULATION & TRAINING TECHNOLOGY CENTER RDRL HRT COL M CLARKE RDRL HRT I MARTINEZ RDRL HRT T R SOTTILARE RDRL HRT B N FINKELSTEIN RDRL HRT G A RODRIGUEZ RDRL HRT I J HART RDRL HRT M C METEVIER RDRL HRT S B PETTIT 12423 RESEARCH PARKWAY ORLANDO FL 32826
1 (PDF)	ARMY RSCH LABORATORY – HRED (PDF) HQ USASOC RDRL HRM CN R SPENCER BLDG E2929 DESERT STORM DRIVE FORT BRAGG NC 28310
1 (PDF)	ARMY G1 DAPE MR B KNAPP 300 ARMY PENTAGON RM 2C489 WASHINGTON DC 20310-0300
	ABERDEEN PROVING GROUND
13 (PDF)	DIR USARL RDRL HR L ALLENDER P FRANASZCZUK C COSENZO RDRL HRM P SAVAGE-KNEPSHIELD RDRL HRM AL C PAULILLO RDRL HRM B C SAMMS RDRL HRM C L GARRETT RDRL HRS J LOCKETT RDRL HRS B M LAFIANDRA RDRL HRS C W HAIRSTON K MCDOWELL RDRL HRS D B AMREIN RDRL HRS E D HEADLEY